Debugging avanzato per Linux embedded

Una panoramica sulle soluzioni disponibili per il debug di sistemi Linux multicore e Android

Marco Ferrario
Lauterbach Italia

Fig. 1 - La soluzione Lauterbach per il debug di sistemi Linux embedded e Android

L'esigenza di adottare architetture di CPU multicore nasce dal tentativo di risolvere i problemi di consumo energetico e conseguente dissipazione di calore, che si presentano quando si aumenta la frequenza di lavoro di una CPU a singolo core. Inoltre la tendenza tecnologica a ridurre la geometria di un chip aumenta sempre più i problemi di progetto, dovuti al manifestarsi di fenomeni parassiti (effetti capacitivi, correnti di dispersione, e così via).

Una prima risposta a questi problemi consiste nel definire nuove architetture di CPU, dotate per esempio di superpipeline oppure di tipo superscalare. Nel primo caso si riduce ogni stadio della pipeline in unità più piccole. Nel secondo caso si aumenta il numero di pipeline facendole lavorare in parallelo. Questo tipo di soluzione comporta una serie di problemi, in particolare circa le condizioni di stallo della pipeline e della complessità della sincronizzazione.

Un'altra tendenza è quella di aumentare le unità logico-aritmetiche (ALU), rendendo possibile l'elaborazione contemporanea di più istruzioni, ciascuna delle quali opera su dati diversi. Si parla in questo caso di architettura MIMD (multiple instruction multiple data). Un'architettura MIMD a singolo core risulta comunque molto complessa a livello progettuale.

La soluzione oggi comunemente più adottata per risolvere i problemi sopra citati è quella di sviluppare vere e proprie architetture multicore, realizzando quindi diverse unità più piccole al posto di una sola più grande. In questo caso la complessità di progetto si sposta dal singolo core al sistema di comunicazione e di bilanciamento del carico di elaborazione dei core.

Dal punto di vista energetico, la disponibilità di più core permette anche lo spegnimento temporaneo delle unità meno attive, che risulta molto più semplice rispetto allo spegnimento di singoli circuiti di un solo core.

Inoltre i core di una CPU possono anche essere realizzati in modo asimmetrico, dedicando ogni singolo core a un compito specifico. Non è raro il caso in cui una CPU multicore sia composta da un'unità microcontrollore e da un DSP.

Nei casi in cui il bilanciamento del carico di lavoro non sia realizzato a livello hardware, la potenza elaborativa delle architetture multicore può essere opportunamente sfruttata solo se esiste un supporto da parte di un sistema operativo. Il multiprocessing può quindi essere classificato in due modi: sono possibili soluzioni asimmetriche (AMP, asymmetrical multiprocessing) in cui l'assegnamento dei task a un singolo core è definito univocamente in fase progettuale; oppure soluzioni simmetriche (SMP, symme-

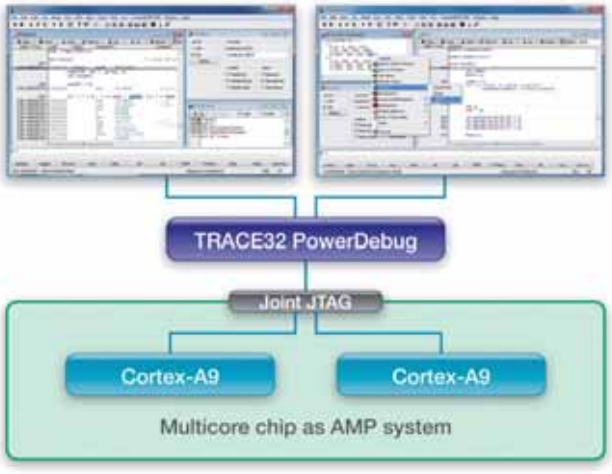

Fig. 2 - Per il debug di sistemi AMP viene avviata una singola istanza di TRACE32 per ogni core

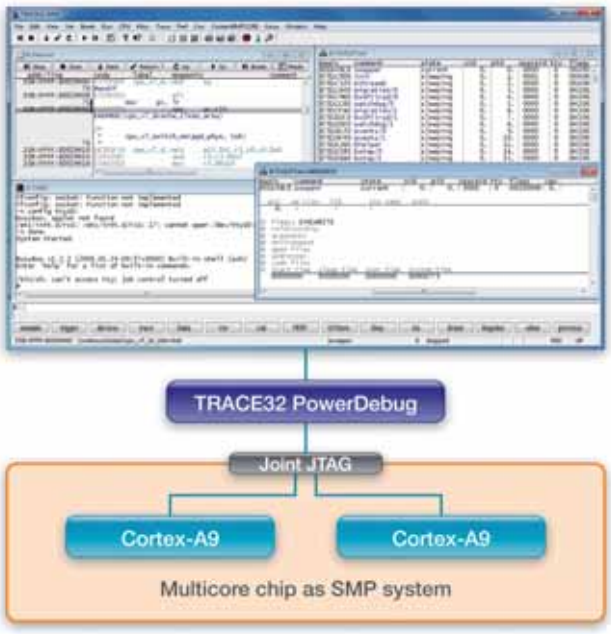

Fig. 3 - Per il debug di sistemi SMP una singola istanza di TRACE32 controlla tutti i core

trical multiprocessing) in cui ogni task è assegnato a un core dinamicamente, da parte di un sistema operativo SMP, sulla base di politiche di assegnamento definite nel sistema operativo stesso. In questo caso tutti i core devono essere dello stesso tipo.

Debug AMP e SMP

Lauterbach (Fig. 1) supporta il debug di sistemi AMP mediante istanze separate del software TRACE32, una per ogni core (Fig. 2). Ciò consente di gestire opportunamente CPU contenenti diverse architetture di core. È tuttavia possibile fermare o avviare contemporaneamente le applicazioni in esecuzione sui diversi core, che spesso operano interagendo fra loro. Invece, per il debug di sistemi SMP, la società fornisce una singola istanza del software TRACE32 in grado di controllare tutti i core (Fig. 3). L'interfaccia utente mostra di volta in volta le informazioni d'interesse per il debug di un'applicazione, riconoscendo autonomamente il core su cui è allocata. La visualizzazione può comunque essere spostata su un altro core. L'assegnamento dei breakpoint è risolto posizionandoli su tutti i core, poiché non è possibile sapere a priori su quale core sarà in esecuzione l'applicazione al momento del breakpoint. Quando un core si ferma a un breakpoint, anche gli altri core vengono fermati. Se il programma è riavviato, tutti i core ripartono insieme.

Trace AMP e SMP

Il trace real time di un'applicazione permette di rilevare rapidamente e sistematicamente condizioni di malfunzionamento particolarmente complesse, che si verificano solo in condizioni runtime. È inoltre possibile ottenere informazioni statistiche, come l'analisi di copertura del codice eseguito e, avendo a disposizione anche informazioni sul tempo di esecuzione, il profiling della durata delle singole funzioni, in modo da verificare la corrispondenza con eventuali requisiti temporali. Molti sistemi rendono disponibile anche il trace di dati d'interesse dell'utente, ad esempio l'informazione sul processo correntemente in esecuzione in un sistema operativo multiprocessing. Lauterbach realizza il trace di sistemi AMP in modo indipendente per ogni core. Ogni singola istanza del software TRACE32 permette di visualizzare le informazioni di trace del core corrispondente. È comunque possibile sincronizzare le diverse viste grazie all'utilizzo di un unico riferimento temporale (Fig. 4). Invece, nei sistemi SMP Lauterbach utilizza un'unica memoria di trace condivisa fra tutti i core (Fig. 5). È quindi possibile visualizzare le informazioni di trace contemporaneamente per tutti i core, oppure distinguendo fra i singoli core o addirittura per singoli task (Fig. 6).

Linux awareness

Il riconoscimento delle strutture dati rilevanti di un sistema operativo e dei suoi meccanismi di gestione delle risorse (awareness) permette a un debugger di offrire all'utente funzionalità avanzate di controllo del software. Nel caso di Linux utilizzato su sistemi embedded, i debugger Lauterbach forniscono un'integrazione molto sofisticata con il sistema operativo, in grado di mostrare le condizioni di utilizzo delle principali risorse allocate (Fig. 7). Per garantire queste funzionalità, l'awareness riconosce la configurazione dell'MMU del processore, sapendo che Linux opera in uno spazio di memoria virtuale. Si noti che il meccanismo di demand paging di Linux comporta che un'applicazione possa essere fisicamente non presente in memoria, fintanto che le singole pagine contenenti istruzioni o dati non vengano richieste per l'esecuzione. Il supporto a Linux in Lauterbach permette di effettuare il debug di un processo utente a partire dal suo avvio. Se il processo utilizza librerie dinamiche (shared objects)

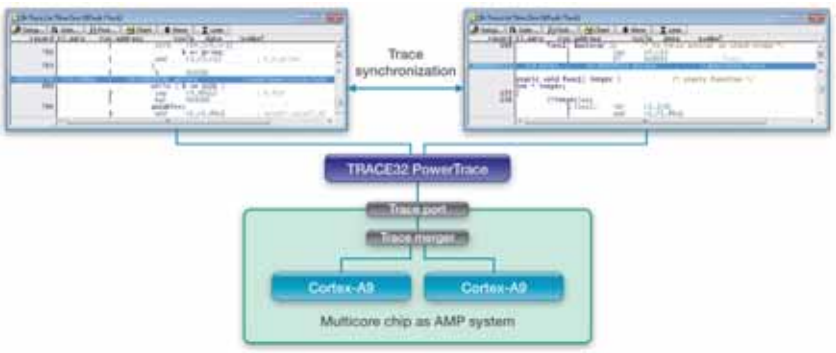

Fig. 4 - Il trace di sistemi AMP viene visualizzato su GUI distinte. È comunque possibile sincronizzare le diverse interfacce

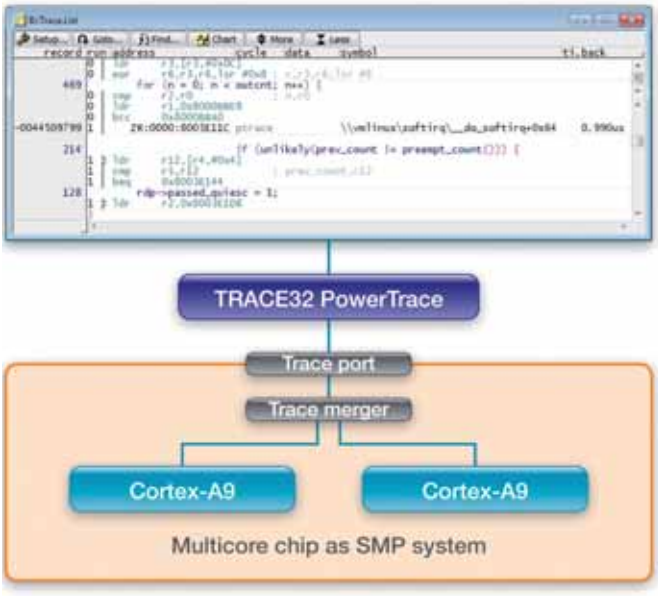

Fig. 5 - Nei sistemi SMP si utilizza una sola shared memory per memorizzare i dati di trace di tutti i core

Linux le carica nello spazio di indirizzamento del processo. Occorre tenere presente che anche le librerie dinamiche vengono caricate da Linux nel momento in cui le loro istruzioni sono utilizzate per la prima volta. Con i debugger Lauterbach è inoltre possibile il debug dei threads che compongono un processo. In questo caso è sufficiente caricare una sola volta le informazioni simboliche associate al processo.

Il kernel di Linux è compilato in modo da consentire il collegamento di moduli aggiuntivi, che possono essere caricati dinamicamente. Con Lauterbach è possibile effettuare il debug di un modulo a partire dalle funzioni di inizializzazione.

È inoltre facilitato il debug delle eccezioni di segmentation violation, mediante l'assegnamento dei breakpoint necessari per identificare l'eccezione e la possibilità di caricare temporaneamente i registri del processore con lo stato macchina che ha provocato l'eccezione, così da facilitare l'analisi del problema. Al momento in cui l'esecuzione viene ripresa, vengono ripristinati i registri originali.

È anche possibile associare dei breakpoint a un singolo task ed eseguire per un singolo task il trace del flusso di programma. Nel caso in cui il processore fornisca funzionalità di trace dei dati, è possibile tracciare il cambio di contesto dei singoli task, con il supporto del sistema operativo.

Android

Android è un sistema operativo open source basato sul kernel Linux. Le applicazioni vengono eseguite mediante una macchina virtuale Java adattata per l'utilizzo su dispositivi mobili, chiamata

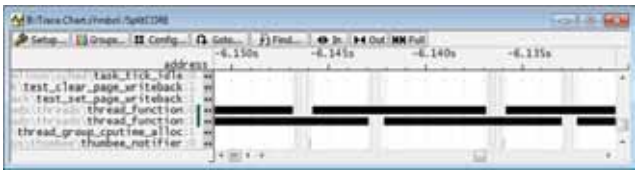

Fig. 6 - È possibile rappresentare in forma grafica i dati di trace, distinguendo in base al core di esecuzione

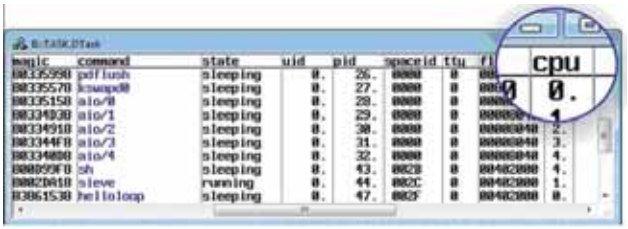

Fig. 7 - L'awareness di Linux permette di esaminare l'utilizzo delle risorse del sistema operativo, come ad esempio i task in esecuzione

Dalvik virtual machine. Il codice di sistema, in esecuzione al di fuori della macchina virtuale, comprendente i servizi di base, i driver e il kernel stesso, viene denominato codice nativo.
Le applicazioni Java/Dalvik possono essere sviluppate e testate con il supporto di un apposito SDK. Anche Eclipse fornisce un plugin dedicato, con il quale è possibile collegarsi via ADB (Android Debug Bridge) a un debug daemon sul target per scopi di debug. In ambito embedded, esistono tuttavia molte situazioni in cui non è sufficiente eseguire il debug della sola applicazione Java. Si tratta ad esempio di casi in cui è necessario modificare dei componenti di sistema, come lo stack di rete, oppure analizzare l'interazione fra un'applicazione e un driver di basso livello. Si pensi anche a casi di analisi post mortem, in cui è necessario studiare lo stato di un sistema che ha appena subìto un crash inatteso, per ricostruirne le cause. In queste situazioni sarebbe preferibile utilizzare un debugger in grado di accedere sia al codice nativo sia al codice Java. Tuttavia il codice Java presenta un problema: non si tratta infatti di un vero e proprio codice macchina, ma di una rappresentazione in linguaggio intermedio, detta bytecode, rappresentabile a sua volta come dati per l'interprete di una macchina virtuale. In altre parole, il codice Java non è altro che una serie di dati forniti in ingresso ad un apposito programma. Dal punto di vista della macchina fisica, si tratta dunque di informazioni diverse rispetto alle istruzioni in codice nativo che la macchina fisica può eseguire. È quindi comprensibile per quali motivi risulti difficile integrare in uno stesso ambiente il debug del codice nativo e il debug del codice Java. Si tratta di due tipi diversi di codice, eseguiti diversamente fra loro e non riconducibili l'uno all'altro in modo elementare. Anche dal punto di vista

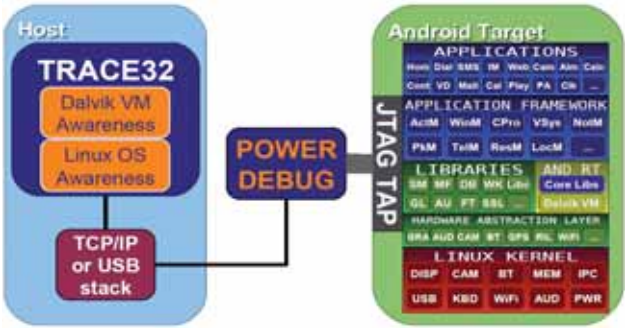

Fig. 8 - I diversi livelli di astrazione supportati dal software TRACE32 permettono il debug di tutte le risorse presenti in un sistema Android

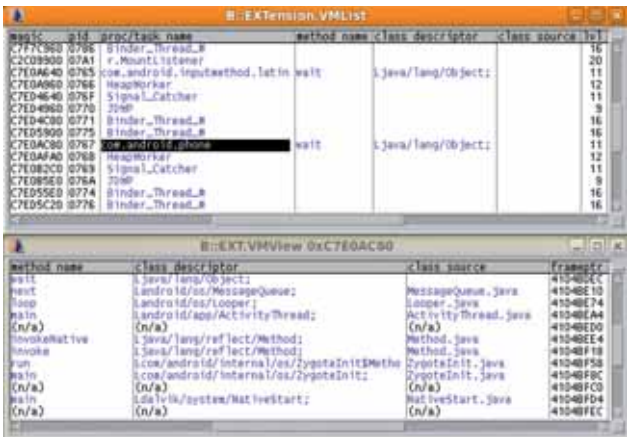

Fig. 9 - Lauterbach supporta l'awareness della Dalvik virtual machine, permettendo il debug delle applicazioni Java

del trace esistono delle differenze. Se infatti il codice nativo può essere tracciato con dei meccanismi convenzionali di trace del flusso di programma, per tracciare un'applicazione Java è necessario avere un sistema in grado di catturare il flusso dei dati.
I debugger Lauterbach forniscono un ambiente che permette il cosiddetto stop mode debugging, ovvero la possibilità di fermare il processore che esegue la macchina fisica, congelando dunque in uno stato ben definito la piattaforma che contiene il sistema operativo e le applicazioni in esecuzione, in modo da consentirne un'analisi tramite il debugger. Ciò richiede un supporto dall'hardware, normalmente ottenuto mediante una porta di accesso JTAG. In questo modo è possibile eseguire il debug del codice nativo, comprendendo anche il kernel per il quale è disponibile uno specifico supporto mediante funzionalità di OS awareness. Per permettere il debug congiunto del codice nativo e delle applicazioni Java, Lauterbach sta sviluppando un'estensione speciale delle funzionalità di awareness, in grado di gestire una macchina virtuale Dalvik (Fig. 8). In tal modo sarà possibile esaminare lo stato dei programmi in esecuzione nella macchina virtuale, con le stesse potenzialità attualmente disponibili per accedere ai programmi in codice nativo (Fig. 9).